Adaptive Algorithm for Minimizing Cloud Task Length with Prediction Errors

Sheng Di, *Member IEEE,* Cho-Li Wang, *Member IEEE,* Franck Cappello, *Member IEEE*

Abstract—Compared to traditional distributed computing like Grid system, it is non-trivial to optimize cloud task's execution performance due to its more constraints like user payment budget and divisible resource demand. In this paper, we analyze in-depth our proposed optimal algorithm minimizing task execution length with divisible resources and payment budget: (1) We derive the upper bound of cloud task length, by taking into account both workload prediction errors and hostload prediction errors. With such state-of-the-art bounds, the worst-case task execution performance is predictable, which can improve the Quality of Service in turn. (2) We design a dynamic version for the algorithm to adapt to the load dynamics over task execution progress, further improving the resource utilization. (3) We rigorously build a cloud prototype over a real cluster environment with 56 virtual machines, and evaluate our algorithm with different levels of resource contention. Cloud users in our cloud system are able to compose various tasks based on off-the-shelf web services. Experiments show that task execution lengths under our algorithm are always close to their theoretical optimal values, even in a competitive situation with limited available resources. We also observe a high level of fair treatment on the resource allocation among all tasks.

Keywords—Algorithm, Cloud Computing, Divisible-Resource Allocation, Convex Optimization, Upper Bound Analysis

1 Introduction

Cloud computing has emerged as a compelling paradigm for the deployment of ease-of-use virtual environment on the Internet. One of typical features in Cloud computing is its pool of instantly accessible virtualized resources that can be dynamically customized, while optimizing the resource utilization.

Traditional task scheduling adopted in distributed systems like Grids assumes discrete resource usage model [1], [2], [3]. The processing ability assigned to a task cannot be customized by users elastically. Such an indivisible resource consumption model with discrete computation units results in a non-trivial problem like binary Integer programming problem, where CPU rates may not be fully utilized.

With virtual machine (VM) resource isolation technology [4], [5], [6], [7], [8], [9], the computational resources could be partitioned and reassembled on demand, creating an avenue to improve resource utilization. In our previous work [10], we proposed an optimal algorithm (namely local optimal allocation algorithm (LOAA)) minimizing a task's execution length, subject to a set of constraints like user's payment budget and host availability states.

In comparison to the previous work, we further make four new contributions in this paper.

- First of all, we extend the problem formulation by taking into account possible execution cost like the extra time in loading VM images. Not only can we prove that LOAA algorithm is still an optimal solution to minimize task length, but we can prove that user payment is also minimized based on task's final wall-clock length.
- We further analyze the upper bound of task execution length when task's workload and host's availability were predicted with errors, which is more in the line with reality. For instance, the *multi-variate polynomial regression* method [14] and Bayes method [15] have been effective in precise workload prediction and hostload prediction respectively, yet they are still suffering inevitable margin of prediction errors like 10%. On the other hand, the flexible resource partitioning of the Cloud systems may definitely result in load dynamics on resource states, and worse still, the collected states are error-prone due to the network propagation delay. The inevitable load prediction errors may significantly affect task's execution in reality. In this paper, we derive the bound of task length for the LOAA algorithm, based on erroneous prediction of task's workload and resource availability, as compared to the theoretically optimal task length with hypothetically accurate information. This is fairly valuable/useful in that users are able to know the worst performance in advance and the resource allocation can be tuned in turn to adapt to user demand based on the bound of task execution length estimated.
- We further extend our algorithm to a dynamic version to adapt to the load dynamics over time. Due to the dependency between the subtasks (or web services) of a task, the resource availability states for a particular subtask may not be forecasted upon

- *S. Di is currently a post-doctor researcher at INRIA (France) and Argonne National Lab (USA), C.L. Wang is with the Department of Computer Science, The University of Hong Kong (Hong Kong), and Franck Cappello is a senior computer scientist at Argonne National Lab (USA).*

the task's initial submission. Accordingly, we extend our algorithm to be a dynamic (or adaptive) version, which can tune the resource allocation at run-time based on task's execution progress and updated resource availability states.

- To demonstrate the practical use of the proposed solution, we built a Cloud prototype, which can optimize the execution performance of user-customized complex web services. By leveraging Xen's credit scheduler [16] and Linux network traffic controller (TC) [17], we evaluate our algorithm based on divisible CPU rates, network and disk bandwidth. Based on a well-known matrix library called *ParallelColt* [18], we emulate hundreds of Cloud tasks involving different types of execution properties (such as CPU-bound, memory-bound and I/O-bound). Experiments confirm our solution is able to effectively restrict task's execution length, and the experimental results exhibit close to the theoretical upper bound derived with load prediction errors. We also implement another heuristic (called LPRPS) which intuitively maximizes the load-price-ratio proportionally. Our optimal solution significantly outperforms this heuristic by 4 times with respect to the average execution stretch.

The remainder of the paper is organized as follows. In Section 2, we formulate the cloud resource allocation issue as a convex optimization problem aiming to minimize task length with divisible resource fractions and a set of constraints [12]. In Section 3, we outline the task processing procedure and describe the LOAA algorithm. We prove that LOAA algorithm can also minimize user payments meanwhile based on tasks' final real wall-clock lengths. In Section 4, we derive the upper bound of task execution length considering prediction errors on task workloads and resource availability, as against to the result under the hypothetically precise prediction. In Section 5, we extend our algorithm to adapt to the volatile states of the system with multiple web services deployed. We present the experimental results generated over a real-cluster environment in Section 6. We discuss the related works in Section 7 and conclude with a vision of the future work in Section 8.

2 Problem Formulation

Our system follows a popular data center model (or server/client model) to process cloud user requests. The cloud server is responsible for collecting dynamic availability states of managed nodes and customizing virtual machines based on users' various demands. A task execution can be split into three steps: (1) Select a qualified physical node. (2) The task is running in an individual VM instance, whose resources are customized on demand by virtual machine monitor (VMM), a.k.a., hypervisor. (3) Send computational results to users.

Different task executions are likely of different execution patterns in that they need multiple types of resources (or execution dimensions). For example, one task execution may be split into multiple steps, each demanding a different CPU rate or I/O bandwidth.

Suppose there are n physical nodes (denoted p_i, where $1 \leq i \leq n$). For any particular task requiring R execution dimensions (e.g., R phases each with different types of resources), we denote the complete set of execution dimensions by Π and denote p_i's capacity vector on multiple dimensions by $\boldsymbol{c}(p_i)=(c_1(p_i), c_2(p_i), \cdots, c_R(p_i))^T$. We use d_k to denote the kth execution dimension (i.e., some specific type of resource).

A task assigned to node p_i is denoted by t_{ij}, where $1 \leq j \leq m_i$, and m_i refers to the number of tasks assigned to p_i. The *workload vector* of the task t_{ij} to process in multiple dimensions is denoted by $\boldsymbol{l}(t_{ij})=(l_1(t_{ij}), l_2(t_{ij}), \cdots, l_R(t_{ij}))^T$. Hence, t_{ij} needs a resource vector to complete its workloads, and we denote such a vector as $\boldsymbol{r}(t_{ij})=(r_1(t_{ij}), r_2(t_{ij}), \cdots, r_R(t_{ij}))^T$, where $r_k(t_{ij})$ (k=1,2,$\cdots$,R) refers to the resource fraction split from t_{ij}'s assigned node p_i. Node p_i's resource availability vector on multiple dimensions (denoted $\boldsymbol{a}(p_i)$) is calculated by $\boldsymbol{c}(p_i) - \sum_{t_{ij}\, running\, on\, p_i} \boldsymbol{r}(t_{ij})$. The resource to be allocated to a task t' must conform to Inequality (4), where $\preceq$ means componentwise inequality between two vectors. We give an example to further illustrate task workload vector ($\boldsymbol{l}(t_{ij})$), allocated resource vector ($\boldsymbol{r}(t_{ij})$) and resource availability vector ($\boldsymbol{a}(p_i)$). Suppose a task's execution is determined by three types of resources including CPU rate, diskI/O rate, and network bandwidth rate. It then has three types of workloads to process - an amount of computation to be performed by CPU, an amount of data to read from disk, and an amount of data to transmit through network. That is, The task's workload is a three-dimensional vector, and each element of the vector must be no greater than the corresponding available resource rate of the selected execution node.

In our model, t_{ij}'s execution length (or execution time) is defined as $\boldsymbol{l}(t_{ij})^T \cdot \boldsymbol{r}(t_{ij})^{-1} + \triangle$ (Equation (2)), where $\boldsymbol{r}(t_{ij})^{-1}=(r_1^{-1}(t_{ij}), r_2^{-1}(t_{ij}), \cdots, r_R^{-1}(t_{ij}))^T$ and $\triangle$ implies a constant extra cost (such as VM-loading time). Such a definition of execution time specifies a broad set of applications, each of which needs a series of phases to process (or mixed non-overlapped executions on various types of resources). For example, computing the matrix formula $(A_{m\times n} \cdot A_{n\times m})^k \boldsymbol{x} = \boldsymbol{v}_{m\times 1}$ could be split into 5 phases - loading matrix from disk, computing the product $C_{m\times m} = A_{m\times n} \cdot A_{n\times m}$, computing matrix-power $C_{m\times m}^k$, solving $C_{m\times m}\boldsymbol{x} = \boldsymbol{v}_{m\times 1}$, and storing $\boldsymbol{x}$ onto disk.

The R types of resources (or R execution dimensions) are associated with a price vector denoted as $\boldsymbol{b}(p_i)=(b_1(p_i), b_2(p_i), \cdots, b_R(p_i))^T$. Let $b_k(p_i)$ ($1 \leq k \leq R$) denote the per-time-unit price the consumers need to pay for the resource consumption of p_i at kth execution dimension. Then, t_{ij}'s total payment $\rho(t_{ij}, \Delta t)$ can be calculated via Equation (1), where Δt refers to t_{ij}'s execution period on p_i.

$$\rho(t_{ij}, \Delta t) = \Delta t \cdot \boldsymbol{b}(p_i)^T \cdot \boldsymbol{r}(t_{ij}) \tag{1}$$

We argue that it is non-trivial to precisely predict task execution length and quantify the resource consumption at each dimension, thus it is inviable for users to forecast their total payment in advance. Hence, we adopt the pay-by-reserve policy, in which the users could get the reserved resources for the task execution. The users feel happy as long as the per-time-unit rate is always within an acceptable budget (denoted $B(t_{ij})$), i.e., Formula (3). Such a payment policy is widely adopted by Cloud systems like OpenPEX [19], and also recommended by Amazon EC2 for cost modeling research [20].

In the following text, we might omit the symbols t_{ij} and p_i if thus would not cause ambiguity (especially when a task is already determined). For instance, $l_k(t_{ij})$, $\boldsymbol{r}(t_{ij})$, $b_k(p_i)$, $\boldsymbol{a}(p_i)$ and $B(t_{ij})$ may be substituted by l_k, $\boldsymbol{r}$, b_k, $\boldsymbol{a}$, and B respectively.

Finally, the resource allocation problem is modeled as a convex optimization problem: for a submitted task t' with its workload vector $\boldsymbol{l}(t')$, how to minimize its execution length (i.e., Equation (2)) subject to constraints (3) and (4), where p_e is selected from among all managed hosts and $\boldsymbol{r}(t', p_e)$ means executing t' on p_e).

$$f(\boldsymbol{r}(t', p_e)) = \boldsymbol{l}^T(t') \cdot \boldsymbol{r}^{-1}(t') + \triangle = \sum_{i=1}^{R} \frac{l_i}{r_i} + \triangle \tag{2}$$

$$\boldsymbol{b}(p_e)^T \cdot \boldsymbol{r}(t') \leq B(t'),\ where\ p_e\ is\ execution\ node \tag{3}$$

$$\boldsymbol{r}(t') \preceq \boldsymbol{a}(p_e) \tag{4}$$

We summarize the key notations in Table 1.

TABLE 1: Key Notations

Notation	Description
n	number of nodes
p_i (or p_e)	a physical node, where i = 1,2,$\cdots$,n
d_k	the kth execution dimension (i.e., some type of resource)
$\boldsymbol{c}(p_i)$	capacity vector of node p_i
$\boldsymbol{b}(p_i)$	price vector of p_i on multiple dimensions
t_{ij}	jth task scheduled on p_i
$\boldsymbol{l}(t_{ij})$	workload vector of t_{ij}
$\boldsymbol{r}(t_{ij})$	the resource fraction vector allocated to t_{ij}
$\boldsymbol{r}^{-1}(t_{ij})$	=$(r_1^{-1}(t_{ij}), r_2^{-1}(t_{ij}), \cdots, r_R^{-1}(t_{ij}))^T$
$\triangle$	a constant extra cost in task execution, e.g, VM-loading time
$B(t_{ij})$	budget of t_{ij}'s user (evaluated by per-time-unit)
$\rho(t_{ij}, \Delta t)$	payment of the user on executing t_{ij}
$\boldsymbol{r}(t_{ij}, p_e)$	the resource vector split from p_e for task t_{ij}
$\boldsymbol{a}(p_i)$	availability vector of p_i $(=(a_1(p_d), a_2(p_d), \cdots, a_R(p_d))^T$

3 Optimal Divisible-Resource Allocation (ODRA)

We outline the pseudo-code of the skeleton algorithm in Algorithm 1, which describes how to select nodes for the specific task t'. The input list contains two parts, resource information and task information. The former includes the candidate node set S=$\{p_1,p_2,\cdots,p_n\}$, price vector set P=$\{\boldsymbol{b}_1,\boldsymbol{b}_2,\cdots,\boldsymbol{b}_n\}$ and availability vector set A=$\{\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_n\}$; the latter includes t''s budget $B(t')$ and its predicted workload vector $\boldsymbol{l}(t')$.

Algorithm 1 Skeleton of ODRA Algorithm

```
Input: $S$, $P$, $A$, $B(t')$, $\boldsymbol{l}(t')$;
Output: execution node $p_e$, $\boldsymbol{r}^*(t')$
1: $p_e$=$p_1$;
2: for (each node $p_i$ in $S$) do
3:     $\boldsymbol{r}^*(t',p_i)$ = LOAA($B(t')$,$\boldsymbol{l}(t')$,$\boldsymbol{a}_i$,$p_i$); /*Calculate the optimal resource allocation for task $t'$ running on node $p_i$.*/
4:     Estimate $f(p_i)$ based on $\boldsymbol{r}^*(t', p_i)$ /*Presume $t'$'s time*/
5:     if ($f(p_i)$<$f(p_e)$) then
6:         $p_e$=$p_i$;
7:         $\boldsymbol{r}^*(t')$=$\boldsymbol{r}^*(t', p_i)$;
8:     end if
9: end for
```

According to Algorithm 1, Line 3 aims to perform a Local Optimal divisible-resource Allocation Algorithm (*LOAA*) on each physical node with low time complexity (to be described in Algorithm 2). This is the most critical step in that the rest part (Line 4$\sim$7) just selects the node which can achieve the shortest execution length.

In the following text, Theorem 1 presents the optimal resource fraction without the constraint (4), and Algorithm 2's output is the optimal resource fraction vector with taking into account the constraint (4). We prove its optimality using Theorem 2 and Theorem 3.

Theorem 1: In order to minimize $f(\boldsymbol{r}(t_{ij}))$ subject to the constraint (3), t_{ij}'s optimal received resource vector $\boldsymbol{r}^{(*)}(t_{ij})$ is shown as Equation (5), where k=1, 2, $\cdots$, R. (Note that $\boldsymbol{r}^{(*)}(t_{ij})$ is not subject to Inequality (4), unlike the notation $\boldsymbol{r}^*(t_{ij})$ that relies on Inequality (4).)

$$r_k^{(*)}(t_{ij}) = \frac{\sqrt{l_k(t_{ij}) \big/ b_k(p_i)}}{\sum\limits_{k=1}^{R} \sqrt{l_k(t_{ij}) b_k(p_i)}} \cdot B(t_{ij}) \tag{5}$$

Proof: We will first prove that the target function $f(\boldsymbol{r})$ formulated in Formula (2) is convex, and then find the optimal $\boldsymbol{r}^{(*)}$ via convex optimization.

Since $\frac{\partial^2 f(\boldsymbol{r})}{\partial r_k}$=$2\frac{l_k}{r_k^3}$>0, $f(\boldsymbol{r})$ is convex with a minimum extreme point. Then, the target *Lagrangian* function can be defined as Equation (6) and λ is the Lagrangian multiplier.

$$F(\boldsymbol{r}) = \sum_{k=1}^{R} \frac{l_k}{r_k} + \triangle + \lambda\left(\sum_{k=1}^{R} b_k \cdot r_k - B\right) \tag{6}$$

Let $\frac{\partial F(\boldsymbol{r})}{\partial r_k}$=0, then we could get Equation (7), where k = 1, 2, $\cdots$, R.

$$l_k / r_k^2 = \lambda b_k \tag{7}$$

From Equation (7), we could derive Equation (8).

$$r_1 : r_2 : \cdots : r_R = \sqrt{\frac{l_1}{b_1}} : \sqrt{\frac{l_2}{b_2}} : \cdots : \sqrt{\frac{l_R}{b_R}} \tag{8}$$

In order to minimize total execution time $f(\boldsymbol{r})$, the optimal resource vector $\boldsymbol{r}^{(*)}$ should make $\boldsymbol{b}(p_i)^T \cdot \boldsymbol{r}(t_{ij})$ equal to $B(t_{ij})$. By combining this equation with Equation (8), we can get Equation (5). □

Remark: By combining the constraint (4), $\boldsymbol{r}^{(*)}$ based on Equation (5) is right the optimal solution as long as $\boldsymbol{r}^{(*)} \preceq \boldsymbol{a}(p_i)$. However, if $\boldsymbol{r}^{(*)}$ does not fully satisfy the constraint (4) (i.e., $\exists\ k$: $r_k^{(*)} > a_k(p_i)$), $\boldsymbol{r}^{(*)}$ should not be a viable solution. Hence, we propose an efficient algorithm (Algorithm 2) to find the optimal solution subject to the constraint (4) with the provable time complexity $O(R^2)$.

Definition 1: Given a specific budget C for task t_{ij}'s execution in a subset $\Gamma(\subseteq\Pi$, i.e., subset of the execution dimension set), CO-STEP(Γ, C) is defined as the procedure in computing the convex-optimal solution of minimizing $f(\boldsymbol{r}_\Gamma(t_{ij}))$ subject to the constraint (9) (similar to the proof of Theorem 1), where $\boldsymbol{r}_\Gamma(t_{ij})$ and $\boldsymbol{b}_\Gamma^T(p_i)$ denote the resource fractions gained by t_{ij} and the price vector assigned by p_i w.r.t. Γ respectively.

$$\boldsymbol{b}_\Gamma^T(p_i) \cdot \boldsymbol{r}_\Gamma(t_{ij}) \le C, \quad \textit{where } C \textit{ is a constant.} \tag{9}$$

Algorithm 2 (namely *LOAA*) is devised for minimizing $f(\boldsymbol{r}(t_{ij}))$ subject to the constraints (3) and (4).

Algorithm 2 LOCAL OPTIMAL ALLOCATION ALGORITHM

```
function name: LOAA($\Pi$, $B(t_{ij})$, $\boldsymbol{l}(t_{ij})$, $\boldsymbol{b}(p_i)$, $\boldsymbol{a}(p_i)$);
Input:  $\Pi$: the execution dimension set (universe)
        $B(t_{ij})$: $t_{ij}$'s budget
        $\boldsymbol{l}(t_{ij})$: $t_{ij}$'s predicted workload vector
        $\boldsymbol{b}(p_i)$: execution node $p_i$'s price vector
        $\boldsymbol{a}(p_i)$: execution node $p_i$'s availability vector
Output: $\boldsymbol{r}^*(t_{ij})$: $t_{ij}$'s optimal resource allocation vector on $p_i$
1: $\Gamma = \Pi$, $C = B(t_{ij})$, $\boldsymbol{r}^* = \varnothing$ (empty set);
2: repeat
3:     $\boldsymbol{r}_\Gamma^{(*)}$ = CO-STEP($\Gamma$,$C$); /*Compute optimal $r^{(*)}$ based on $\Gamma$ with unbounded capacity assumption*/
4:     $\Omega=\{d_k | d_k\in\Gamma\ \&\ r_k^{(*)}>a_k\}$;/*select elements violating constraint (4)*/
5:     $\Gamma = \Gamma\backslash\Omega$; /*$\Gamma$ takes away $\Omega$*/
6:     $C = C - \sum_{d_k\in\Omega}(b_k \cdot a_k)$; /*Update $C$*/
7:     $\boldsymbol{r}^* = \boldsymbol{r}^*\cup\{r_k^* = a_k \mid d_k\in\Omega\ \&\ a_k$ is $d_k$'s upper bound$\}$;
8: until ($\Omega = \varnothing$);
9: $\boldsymbol{r}^* = \boldsymbol{r}^* \cup \boldsymbol{r}_\Gamma^{(*)}$;
```

In this algorithm, line 3 executes CO-STEP(Γ,C) in order to find the optimal $\boldsymbol{r}_\Gamma^{(*)}$, without considering the constraint (4). If $\boldsymbol{r}_\Gamma^{(*)}$ happens to completely satisfy the constraint (4) (i.e., Ω=$\varnothing$), then it is the final result. Otherwise, the resource fractions (r_k) that violate the constraint (4) will be set to the upper bounds (i.e., a_k) and the corresponding dimensions (i.e., Ω) will be taken away from Γ, then, $C = C - \sum_{d_k\in\Omega}(b_k \cdot a_k)$ for the remaining dimensions. The process will go on until all the remaining computed optimal resource fractions satisfy the constraint (4). Since the time complexity of CO-STEP(Γ,C) is $O(|\Gamma|)$, the number of computation steps of Algorithm 2 in the worst case is $\sum_{i=0}^{R-1}(R-i)$, thus the time complexity=$O(R^2)$.

Theorem 2: Given a submitted task t_{ij} with a workload vector $\boldsymbol{l}(t_{ij})$ and a budget $B(t_{ij})$ and a qualified node p_i with its resource price vector $\boldsymbol{b}(p_i)$, Algorithm 2's output $\boldsymbol{r}^*$ is optimal for minimizing t_{ij}'s execution length (i.e., $f(\boldsymbol{r}(t_{ij}))$), subject to the constraint (3) and constraint (4).

We can prove Algorithm 2's output must satisfy the sufficient and necessary conditions of optimal solution. We omit the detailed proof, which can be found in [10].

In this paper, we further prove that the Algorithm 2 can also minimize the **user payment** based on the real task wall-clock length (shown in Theorem 3).

Theorem 3: Denote the real wall-clock length of task t_{ij} as $T_f(t_{ij})$. Given t_{ij}'s deadline is set to $T_f(t_{ij})$, Algorithm 2's output $\boldsymbol{r}^*$ is the optimal solution that minimizes this user's payment, subject to t_{ij}'s workload vector $\boldsymbol{l}(t_{ij})$ and node p_i's price vector $\boldsymbol{b}(p_i)$.

Proof: We denote task t_{ij}'s allocated resource vector as $\boldsymbol{r}^*$, then it must satisfy Equation (10) and Equation (11), where B is task's budget used in Algorithm 2.

$$f(\boldsymbol{r}^*(t_{ij})) = \sum\nolimits_{i=1}^{R} \frac{l_i}{r_i^*} = T_f(t_{ij}) \tag{10}$$

$$\sum\nolimits_{i=1}^{R} b_i r_i^* = B_f \le B \tag{11}$$

If Theorem 3 does not hold, there must exist a resource allocation $\boldsymbol{r}'(\neq \boldsymbol{r}^*)$ for running t_{ij} on p_i, simultaneously satisfying $\sum\limits_{i=1}^{R} \frac{l_i}{r_i'} = T_f(t_{ij})$ and Inequality (12).

$$\sum\nolimits_{i=1}^{R} b_i r_i' < B_f \tag{12}$$

Inequality (12) implies that there must exist a sufficiently tiny Δr>0, such that the new resource allocation $\{r_1'+\Delta r,\ r_2',\ r_3',\ \cdots,\ r_R'\}$ also satisfies Inequality (12), yet its execution length (i.e., $\frac{l_1}{r_1'+\Delta r} + \sum_{i=2}^{R}\frac{l_i}{r_i'}$) will be smaller than $f(\boldsymbol{r}^*(t_{ij}))$, which contradicts to the fact that $f(\boldsymbol{r}^*(t_{ij}))$ is already minimized proved by Theorem 2. □

4 OPTIMALITY ANALYSIS WITH LOAD PREDICTION ERRORS

4.1 Problem Description

Although Algorithm 2 is proved optimal, such optimality is subject to both task workload ($\boldsymbol{l}$) and host load ($\boldsymbol{a}$) can be precisely given or predicted. In reality, task's workload may not be precisely predicted by ordinary users. Users' preferences or requirements on resources tend to be qualitative as users' knowledge on resource specification is limited. On the other hand, the host availability states aggregated may also be inaccurate because of the fast changing hostload states. The communication delay and unfledged resource state estimation technologies are other sources of potential errors.

In this section, we analyze Algorithm 2 under the erroneous workload predictions and host availability prediction (i.e., host load prediction). We derive the upper bound of task execution length for Algorithm 2 with such two prediction errors.

Definition 2: Suppose a task t_{ij}'s workload vector is predicted as $\boldsymbol{l}'(t_{ij})$ while the real workload vector is

$\boldsymbol{l}(t_{ij})$. In our analysis, $\boldsymbol{l}'(t_{ij})$ is assumed to satisfy Inequality (13), where α and β are the lower bound and upper bound of the workload prediction ratio.

$$\alpha \leq \frac{l'_k(t_{ij})}{l_k(t_{ij})} \leq \beta,\ k = 1, 2, \cdots R \tag{13}$$

We give an example to illustrate the above definition. Suppose the task t_{ij}'s real workload vector always ranges in [125, 1000] single-core-length, and the workload vector $\boldsymbol{l}'(t_{ij})$ used by Algorithm 2 is based on the history of the task's execution. Each element $l'_k(t_{ij})$ ($k = 1, 2, \cdots R$) will be set to 250 if the corresponding true workload fluctuates in [125, 500] and set to 750 if the true workload ranges in (500, 1000]. Then, we could get Inequality (14) below, where α=$\frac{250}{500}$=0.5 and β=$\frac{250}{125}$=2.

$$0.5 \leq \frac{l'_k(t_{ij})}{l_k(t_{ij})} \leq 2,\ k = 1, 2, \cdots R \tag{14}$$

It is obvious that with the inaccurate prediction of task's workload, the output of Algorithm 2 will definitely be skewed from the result with accurate information. Hence, one question is how far the output produced by Algorithm 2 based on $\boldsymbol{l}'(t_{ij})$ would be away to the ideal result based on $\boldsymbol{l}(t_{ij})$.

On the other hand, the inaccurate information about host availability states (i.e., vector $\boldsymbol{a}(p_i)$) may also impact the optimality of the algorithm's output. Accordingly, the other question is "what is the worst case on task execution length under the resource allocation with host availability prediction errors". For simplicity, we take into account the situation with $\boldsymbol{a}' \preceq \boldsymbol{a}$ as shown in Inequality (15), where the collected host availability vector $\boldsymbol{a}'$ is denoted as $(a'_1, a'_2, \cdots, a'_R)^T$, as compared to the true host availability vector $\boldsymbol{a} = (a_1, a_2, \cdots, a_R)^T$. In fact, if $\boldsymbol{a}' \succ \boldsymbol{a}$, Algorithm 2's output will definitely be no worse than that with $\boldsymbol{a}$ (note that our focus is on how much the inaccurate availability state would degrade the Algorithm 2's output).

$$\gamma \cdot \boldsymbol{a} \preceq \boldsymbol{a}' \preceq \boldsymbol{a},\ \gamma\ is\ a\ constant \tag{15}$$

All in all, our objective is to derive an upper bound of task execution length for Algorithm 2 based on erroneous information (i.e., Inequality (13) & Inequality (15)).

4.2 Upper Bound of Task Execution Length

In this section, we analyze the upper bound of task execution length with possibly erroneous information (about both workload and hostload) described above. We first discuss the situation with erroneous workload prediction yet with correct host availability information, in Theorem 4 and Theorem 5. Then, we further take into account the situation with both workload prediction errors and host availability prediction errors.

4.2.1 Analysis with Workload Prediction Errors and Precise Host Availability Information

For the simplicity of description, we denote $\boldsymbol{r}^*_E$ (=(r^*_{E1}, r^*_{E2}, $\cdots$, r^*_{ER})^T) and f^*_E (=$\sum_{k=1}^R \frac{l_k}{r^*_{Ek}}$+$\triangle$) as the output of Algorithm 2 with the workload prediction errors and the corresponding execution length respectively, and E indicates "Estimation with error". Similarly, we denote $\boldsymbol{r}^*_I$ (=(r^*_{I1}, r^*_{I2}, $\cdots$, r^*_{IR})^T) and f^*_I (=$\sum_{k=1}^R \frac{l_k}{r^*_{Ik}}$ + $\triangle$) as the output with accurate workload vector and the corresponding execution length, respectively, and I indicates "Ideal case". Hence, our objective is to determine the upper bound of $\frac{f^*_E}{f^*_I}$, a.k.a., *approximation ratio*.

Denote $\boldsymbol{r}^{(*)}_E$ the optimal resource allocation with unbounded resource capacities. The output of Algorithm 2 could be split into two situations:

- case 1: $\boldsymbol{r}^*_E(t_{ij})$ = $\boldsymbol{r}^{(*)}_E(t_{ij})$.
- case 2: $\boldsymbol{r}^*_E(t_{ij}) \neq \boldsymbol{r}^{(*)}_E(t_{ij})$.

The first situation indicates that in terms of the skewed workload prediction, all of the resource fractions calculated by the initial CO-STEP in Algorithm 2 happen to be no greater than the corresponding resource capacities. That is, the output of the first-round CO-STEP complies with the Inequality (16).

$$\boldsymbol{r}^{(*)}_E(t_{ij}) \preceq \boldsymbol{a}(p_i) \tag{16}$$

In contrast, the second situation means that the initial CO-STEP cannot fulfill the above condition, and the optimal allocation cannot be found until a few more adjustment steps (line 4 $\sim$ line 7 of Algorithm 2).

We first derive the upper bound of task t_{ij}'s execution length for the first case (i.e., Theorem 4), and then discuss the upper bound (i.e., Theorem 5) for the second one.

Theorem 4: **Given** a task t_{ij} with a budget $B(t_{ij})$, a node p_i whose resource price vector is $\boldsymbol{b}(p_i)$, and a inaccurately estimated workload vector $\boldsymbol{l}'(t_{ij})$ subject to Inequality (13), **then** the tight upper bound of t_{ij}'s execution length under the resource allocation $\boldsymbol{r}^{(*)}_E$ conforms to Inequality (17).

$$\frac{f^{(*)}_E}{f^*_I} \leq \sqrt{\frac{\beta}{\alpha}} \tag{17}$$

Main idea of proof: It is obvious that $\boldsymbol{r}^{(*)}_I$ must be no worse than $\boldsymbol{r}^*_I$ (i.e., $f^{(*)}_I \leq f^*_I$ must always hold), because $\boldsymbol{r}^*_I$ is with more constraints. Thus, we could get the final conclusion as long as we can prove Inequality (18).

$$\frac{f^{(*)}_E}{f^{(*)}_I} \leq \sqrt{\frac{\beta}{\alpha}} \tag{18}$$

Proof: Detailed proof can be found in the corresponding conference paper [13]. □

Theorem 5: **Given** a task t_{ij} with a budget B, a resource node whose available resource vector and price vector are $\boldsymbol{a}$ and $\boldsymbol{b}$ respectively, and an erroneous workload vector $\boldsymbol{l}'$ subject to Inequality (13), **then**, the tight upper bound of t_{ij}'s execution length with resource allocation $\boldsymbol{r}^*_E$ conforms to Inequality (19), where Ω refers to the set of dimensions constructed at Line 4 of Algorithm 2 performed with the inaccurate workload vector ($\boldsymbol{l}'$) and r^*_{Ii} is the optimal allocated resource fraction outputted when using the real workload vector (i.e., $\boldsymbol{l}$).

$$\frac{f^*_E}{f^*_I} \leq \theta\sqrt{\frac{\beta}{\alpha}},\ \ where\ \theta = \frac{B - \sum_{di\in\Omega} r^*_{Ii}b_i}{B - \sum_{di\in\Omega} a_i b_i} \tag{19}$$

Proof: Detailed proof can be found in the corresponding conference paper [13]. □

Remark:

- Note that $\theta \geq 1$ because $r^*_{Ii} \leq a_i$ for any $d_i \in \Omega$.
- If $\Omega = \varnothing$ or $\forall d_i \in \Omega, r^*_{Ii} = a_i$, then $\theta = 1$, conforming to the Inequality (18).
- If $\forall d_i{\in}\Omega{\Rightarrow}r^{(*)}_{Ei}{\geq}a_i$, then, we can further derive Inequality (20) from Inequality (19) and lemma 1.

$$\frac{f^*_E}{f^*_I} \leq \lambda\sqrt{\frac{\beta}{\alpha}},\ \ where\ \lambda = \frac{B - \sqrt{\frac{\alpha}{\beta}}\sum_{di\in\Omega} a_i b_i}{B - \sum_{di\in\Omega} a_i b_i} \tag{20}$$

Lemma 1: If $\forall d_i{\in}\Omega{\Rightarrow}r^{(*)}_{Ei}{\geq}a_i$, Inequality (21) must hold.

$$r^*_{Ii} \geq \sqrt{\frac{\alpha}{\beta}}a_i \tag{21}$$

Proof: For any $d_k \in \Omega$, we take into account two situations by discussing whether $r^{(*)}_{Ik} \geq a_k$.
If $r^{(*)}_{Ik} \geq a_k$, then $r^*_{Ik} = a_k$, because $r^{(*)}_{Ik}$ is calculated by the first-round Co-Step of Algorithm 2. On the other hand, $\alpha{\leq}\beta$. Thus, $r^*_{Ik}{=}a_k \geq \sqrt{\frac{\alpha}{\beta}}a_k$.
If $r^{(*)}_{Ik}{<}a_k$, Algorithm 2 will further recompute r^*_{Ik} using the succeeding looping-round Co-Steps, then $r^*_{Ik}{\geq}\frac{\sqrt{l_k/b_k}}{\sum_{di\notin\Omega'}\sqrt{l_i b_i}}(B - \sum_{di\in\Omega'}\sqrt{a_i b_i})$, where Ω' refers to the set of resource dimensions such that $r^{(*)}_{Ii} \geq a_i$ (i.e., r^*_{Ii} is set to a_i after the first-round Co-Step of Algorithm 2 with accurate information). Accordingly, we can get Formula (22) for any $d_k{\notin}\Omega'$. Note that the second equation in Formula (22) holding is due to the fact that $r^{(*)}_{Ii}$ is a stationary point such that $\frac{\partial f(\boldsymbol{r})}{\partial r_k} = 0, \forall k$.

$$\begin{aligned} r^{(*)}_{Ik} &= \frac{\sqrt{l_k/b_k}}{\sum^R_{i=1}\sqrt{l_i b_i}}B \\ &= \frac{\sqrt{l_k/b_k}}{\sum_{di\notin\Omega'}\sqrt{l_i b_i}}(B - \textstyle\sum_{di\in\Omega'}\sqrt{r^{(*)}_{Ii}b_i}) \\ &\leq \frac{\sqrt{l_k/b_k}}{\sum_{di\notin\Omega'}\sqrt{l_i b_i}}(B - \textstyle\sum_{di\in\Omega'}\sqrt{a_i b_i}) \leq r^*_{Ik} \end{aligned} \tag{22}$$

On the other hand, $\forall\ d_k$ (also including the ones in Ω), we can get Equation (23) and Equation (24) from Equation (5).

$$r^{(*)}_{Ek} = \frac{\sqrt{l'_k/b_k}}{\sum^R_{i=1}\sqrt{l'_k b_k}}B \tag{23}$$

$$r^{(*)}_{Ik} = \frac{\sqrt{l_k/b_k}}{\sum^R_{i=1}\sqrt{l_k b_k}}B \tag{24}$$

Then, we can further derive Inequality (25) by combining Inequality (13).

$$r^{(*)}_{Ek} \leq \sqrt{\frac{\beta}{\alpha}}r^{(*)}_{Ik} \tag{25}$$

Since $\forall d_k{\in}\Omega{\Rightarrow}r^{(*)}_{Ek}{\geq}a_k$ (the assumption of this lemma), we can get that $a_k \leq \sqrt{\frac{\beta}{\alpha}}r^{(*)}_{Ik}$, $\forall d_k \in \Omega$.
Hence, if $\forall d_k{\in}\Omega{\Rightarrow}r^{(*)}_{Ek}{\geq}a_k$, $r^*_{Ik}{\geq}r^{(*)}_{Ik}{\geq}\sqrt{\frac{\alpha}{\beta}}a_k$. □

- If $|\Omega|$=1, suppose its unique element is d_1 without loss of generality, it is obvious that $r^{(*)}_{I1}{\geq}a_1$. Then, we could derive Formula (26) from Inequality (20).

$$\frac{f^*_E}{f^*_I} \leq \lambda\sqrt{\frac{\beta}{\alpha}},\ \ where\ \lambda = \frac{B - \sqrt{\frac{\alpha}{\beta}}a_1 b_1}{B - a_1 b_1} \tag{26}$$

- In more generic situations, where $|\Omega| \geq 2$ and Inequality (21) does not hold, we cannot get the similar conclusion as Inequality (20). This is due to the fact that some resource shares (r^*_{Ii}) in Ω being set equal to their available capacities may be conducted at the second or later Co-Steps of Algorithm 2. That is, we cannot make sure that $r^{(*)}_{Ii} \geq a_i$ for any $d_i{\in}\Omega$, such that Inequality (21) cannot hold. However, we can still get Inequality (27) because of the fact that $r^*_{Ii} \geq 0$.

$$\frac{f^*_E}{f^*_I} \leq \lambda\sqrt{\frac{\beta}{\alpha}},\ \ where\ \lambda = \frac{B}{B - \sum_{di\in\Omega} a_i b_i} \tag{27}$$

4.2.2 Analysis with Task Workload Prediction Errors and Host Availability Prediction Errors

Now, let us focus on the erroneous predictions on both task workload (Inequality (13)) and host availability information (Inequality (15)).

Theorem 6: **Given** a submitted task with a erroneous workload prediction vector $\boldsymbol{l}'$ (subject to Inequality (13)), a payment budget B, and a resource node p_i whose resource price vector and host availability vector are $\boldsymbol{b}$ and $\boldsymbol{a}'$ (subject to Inequality (15)) respectively, **then** with Algorithm 2's output (denoted by $\boldsymbol{r}'^*_E$), the task execution length f'^*_E follows a tight upper bound based on Inequality (28), where γ is defined in Inequality (15).

$$\frac{f'^*_E}{f^*_I} \leq \frac{\theta}{\gamma}\cdot\sqrt{\frac{\beta}{\alpha}} \tag{28}$$

Proof: We denote H as the execution dimension set constructed by Algorithm 2's Line 4 at the "first-round", with the inaccurate state information $\boldsymbol{a}'$. That is, H could be defined as follows:

$$\mathrm{H} = \{d_k | r'^{(*)}_{Ek} \geq a'_k\},\ where\ r'^{(*)}_{Ek} = B\cdot\frac{\sqrt{l_k/b_k}}{\sum^R_{i=1}\sqrt{l_i b_i}} \tag{29}$$

Without loss of generality, we let the sequence numbers of such resource dimensions be 1, 2, $\cdots$, |H|. Then, $f_E'^*$ can be calculated by the Equation (30).

$$f_E'^* = \sum_{i=1}^{|\mathrm{H}|} \frac{l_i}{r_{Ei}'^*} + \sum_{i=|\mathrm{H}|+1}^{R} \frac{l_i}{r_{Ei}'^*} + \triangle \tag{30}$$

We can also write the accurate-information-based algorithm's output (denoted f_E^*) to be Equation (31):

$$f_E^* = \sum_{i=1}^{|\mathrm{H}|} \frac{l_i}{r_{Ei}^*} + \sum_{i=|\mathrm{H}|+1}^{R} \frac{l_i}{r_{Ei}^*} + \triangle \tag{31}$$

As follows, we will prove Inequality (32).

$$\frac{f_E'^*}{f_E^*} \leq \frac{1}{\gamma} \tag{32}$$

In other words, as long as we could prove Inequality (33) and Inequality (34) respectively, then Inequality (32) must hold (Note $\gamma \leq 1$ thus $\Delta \leq \frac{1}{\gamma}\Delta$).

$$\sum_{i=1}^{|\mathrm{H}|} \frac{l_i}{r_{Ei}'^*} \leq \frac{1}{\gamma} \cdot \sum_{i=1}^{|\mathrm{H}|} \frac{l_i}{r_{Ei}^*} \tag{33}$$

$$\sum_{i=|\mathrm{H}|+1}^{R} \frac{l_i}{r_{Ei}'^*} \leq \frac{1}{\gamma} \cdot \sum_{i=|\mathrm{H}|+1}^{R} \frac{l_i}{r_{Ei}^*} \tag{34}$$

It is obvious that $\forall d_i \in \mathrm{H}$ (i.e., i=1,2,$\cdots$,|H|), $r_{Ei}'^* = a_i'$ and $r_{Ei}^* \leq a_i$. With the Inequality (15), we could derive the following inequalities, where i=1,2,$\cdots$,|H| and $\gamma \leq 1$:

$$r_{Ei}'^* = a_i' \geq \gamma \cdot a_i \geq \gamma \cdot r_{Ei}^* \tag{35}$$

Hence, the Inequality (33) should hold.

As follows, we will prove Inequality (34). Equivalently, we will prove $\forall i \geq |\mathrm{H}|+1$, $r_{Ei}'^* \geq \gamma \cdot r_{Ei}^*$ instead.

Recall that H indicates the dimensions accumulated by Algorithm 2's Line 4 at its first-round, thus, $\forall$ $d_i$$\in$H, $r_{Ei}'^{(*)} \geq a_i'$. In addition, since $a_i' \leq a_i$ holds, we could get $r_{Ei}^* \geq a_i' = r_{Ei}'^*$ ($\forall i\leq$ H). Then, we could get Inequality (36).

$$B - \sum_{i=1}^{|\mathrm{H}|} r_{Ei}'^* \cdot b_i \geq B - \sum_{i=1}^{|\mathrm{H}|} r_{Ei}^* \cdot b_i \tag{36}$$

If we perform the convex optimization step (i.e., CO-STEP) on $\{r_{E(|\mathrm{H}|+1)}'^*, r_{E(|\mathrm{H}|+2)}'^*, \cdots, r_{ER}'^*\}$ and $\{r_{E(|H|+1)}^*, r_{E(|H|+2)}^*, \cdots, r_{ER}^*\}$ respectively and denote the output resources are $r_{Ei}'^{[*]}$ and $r_{Ei}^{[*]}$, we could get Equation (37) and Equation (38), where k=H+1, H+2, $\cdots$, R.

$$r_{Ek}'^{[*]}(\mathrm{H}) = (B - \sum_{di\in\mathrm{H}} r_{Ei}'^* b_i) \frac{\sqrt{l_k b_k}}{\sum_{di\in\Pi\backslash\mathrm{H}} \sqrt{l_i b_i}} \tag{37}$$

$$r_{Ek}^{[*]}(\mathrm{H}) = (B - \sum_{di\in\mathrm{H}} r_{Ei}^* b_i) \frac{\sqrt{l_k b_k}}{\sum_{di\in\Pi\backslash\mathrm{H}} \sqrt{l_i b_i}} \tag{38}$$

Then, Inequality (39) must hold.

$$r_{Ei}'^{[*]} \geq r_{Ei}^{[*]},\ where\ i = |\mathrm{H}|+1, \cdots, \mathrm{R} \tag{39}$$

$\forall d_i \in \Pi\backslash\mathrm{H}$ (i.e., $i \geq \mathrm{H}+1$):

If $a_i < r_{Ei}^{[*]}$, we could get $r_{Ei}'^{[*]} \geq r_{Ei}^{[*]} > a_i \geq a_i'$, then $r_{Ei}'^* = a_i'$ and $r_{Ei}^* = a_i$. Hence, $r_{Ei}'^* \geq \gamma \cdot r_{Ei}^*$.

If $a_i \geq r_{Ei}^{[*]}$, then there are two situations as follows:

- If $r_{Ei}'^{[*]} \leq a_i'$, $r_{Ei}'^* = r_{Ei}'^{[*]} \geq r_{Ei}^{[*]} = r_{Ei}^*$.
- If $r_{Ei}'^{[*]} > a_i'$, $r_{Ei}'^* = a_i' \geq \gamma \cdot a_i \geq \gamma \cdot r_{Ei}^*$.

Hence, for any $d_i \in \Pi\backslash\mathrm{H}$, $r_{Ei}'^* \geq \gamma \cdot r_{Ei}^*$, which can be used to derive Inequality (34).

In terms of the Formula (30), (31), (33), (34), we could derive Inequality (32). By combining Theorem 5 and this inequality, we could prove Theorem 6 finally.

In order to prove the tight bound property, we show two cases under which the Inequality (28)'s bounds can be reached.

- If $\boldsymbol{a}' = \gamma \cdot \boldsymbol{a}$ and $\sum_{i=1}^{R} a_i b_i \ll B$, it is easy to see that $f_E'^* = \frac{1}{\gamma} f_I^*$ no matter what values α and β are set to.
- If all the a_i (i=1,2,$\cdots$,R) are extremely big, the resource state's accuracy could be ignored, which means that the resource allocation $\boldsymbol{r}_E^* = \boldsymbol{r}_E^{(*)}$ and $\boldsymbol{r}_I^* = \boldsymbol{r}_I^{(*)}$. In this situation, as long as $\forall i$, $\frac{l_i'}{l_i} = \frac{l_j'}{l_j}$, then, $\boldsymbol{r}_E^* = \boldsymbol{r}_I^*$, which implies the bound is reached.

□

5 Dynamic Optimal Divisible-Resource Allocation (DODRA)

Many of existing systems (such as Google App Engine [21]) leverage PaaS/SaaS architecture, which allows users to customize composite web services (e.g., a task with a set of subtasks connected in series). However, the validity of services depends on the availability of their hosting-nodes, whose states are likely changed over time. Hence, the host availability vector used in the resource allocation for the remaining subtasks is best to be dynamically updated over time. on the other hand, since the subtasks (implemented by off-the-shelf web services) in a task are can be deemed as different execution dimensions, the task execution length is still consistent with a linear target function like Equation (2), where r_i (i=1,2,$\cdots$,R) denotes the corresponding service's processing rate.

Based on the above discussion, we devise a dynamic algorithm, to allocate resources for the tasks each of which is made up of multiple subtasks (or composite web services) connected in series. All of service bodies (or libraries) are deployed on each host machine and classified based on their functions. We denote the submitted task as τ, which is composed by K functions. They constitute a partially-ordered set $\digamma(\tau)$=$(F_1, F_2, \cdots, F_K)^T$. F_i's service price is denoted as b_i (which means the user needs to pay b_i per-processor for executing F_i) and its corresponding service node set S_i is denoted $\{s_{i1}, s_{i2}, \cdots, s_{i|S_i|}\}$. We set F_i's availability a_i = $\max_{\forall j\in[1,|S_i|]}\{a(s_{ij})\}$, i.e., the maximum capacity of the node hosting F_i. Moreover, F_i in $\digamma(\tau)$ is always maintained in a non-decreasing order of their hosting-nodes' availabilities.

In this algorithm, we first compute the optimal resource allocation $\boldsymbol{r}^*$=$(r_{exe1}^*, r_{com1}^*, r_{exe2}^*, r_{exe3}^*, \cdots, r_{exeK}^*)^T$ via Algorithm 2 (line 4~10), by considering the communication cost in transmitting the output of F_1 to F_2 on the

Algorithm 3 DYNAMIC ODRA

```
Input: $\boldsymbol{b}_\tau$=$(b_1,b_2,\cdots,b_K,b_{com})^T$, $B(\tau)$, $\digamma(\tau)$, $S_i$ ($i$=1,2,$\cdots$,K);
Output: $\boldsymbol{r}^*(\tau)$ (a service vector)
 1: $\Gamma$=$\digamma(\tau)$, $C$=$B(\tau)$, $\boldsymbol{r}^*$=$\Phi$ (empty set), $s'$=NULL;
 2: for (each $F_i$ in the partially-ordered $\Gamma$, $i$=1,2,$\cdots$,$K$) do
 3:     Retrieve $\Gamma$'s availability vector $\boldsymbol{a}_\Gamma$=$\{a_i,a_{(i+1)},\cdots,a_K,a_{com}\}$;
 4:     Perform CO-STEP($\Gamma\bigcup\{F_{comi}\}$,$C$), and its output is denoted as:
        $\boldsymbol{r}^{(*)}$=$(r^{(*)}_{exei}, r^{(*)}_{comi}, r^{(*)}_{exe(i+1)}, r^{(*)}_{exe(i+2)} \cdots, r^{(*)}_{exeK})^T$;
 5:     if ($r^{(*)}_{comi} > a_{com}$) then
 6:         $r^*_{comi} = a_{com}$; /*based on Algorithm 2 (LOAA)*/
 7:     end if
 8:     if ($r^{(*)}_{exei} > a_i$) then
 9:         $r^*_{exei} = a_i$; /*based on Algorithm 2 (LOAA)*/
10:     end if
11:     Select node $s_{ij}$ from $S_i$, such that $a(s_{i(j-1)}) < r^*_{exei} \leq a(s_{ij})$;
12:     if ($s' \neq s_{ij}$) then
13:         Perform CO-STEP($\Gamma$,$C$), where $a_i$=$a(s')$;
14:         if ($f(\boldsymbol{r}(\tau,s_{ij}))$<$f(\boldsymbol{r}(\tau,s'))$) then
15:             $s' = s_{ij}$; /*update $s'$ if $s_{ij}$ is better than the old $s'$*/
16:         end if
17:     end if
18:     Execute $F_i$ on the node $s'$;
19:     $\Gamma = \Gamma \backslash F_i$;
20:     $C = C - r^*_i \cdot b_i$;
21:     Sleep until $F_i$ is completed;
22: end for
```

network. As for the communication cost, at each resource allocation round, we just focus on the one between the first two functions (line 4), because the rest ones will be considered at runtime as soon as its preceding task is completed. After computing $\boldsymbol{r}^*$, F_1's execution will be started on a computation node whose CPU rate is closest greater than r^*_{exe1}. Such a design is to save the remaining budget for the task's later usage and save the available resource for other tasks. As F_1 is completed, the program will take away F_1 from Γ and update Γ's real-time availability vector as well as the remaining budget. Based on updated information, $\boldsymbol{r}^*$ will be recomputed (line 4$\sim$11). If the node selected (i.e., s_{ij}) is not the one computing the last function, we need to decide if it is worth changing the execution node based on the transmission overhead (line 12$\sim$17), as the output of previous operation is stored in the last execution node. Then, F_2 will be started and the whole process will continue until all functions are completed.

Algorithm 3 is a dynamic version compared to Algorithm 2, as the resource allocation for each subtask (or each particular execution phase) is performed only when its preceding subtask is finished. This can adapt to the load dynamics to a certain extent. In fact, we further design an *adaptive mechanism* to tune/reallocate the resource vectors for the running subtasks, by leveraging the idle resource fraction released from time to time, further improving resource utilization.

6 PERFORMANCE EVALUATION

6.1 Experimental Setting

We evaluate our Optimal Divisible-Resource Allocation (ODRA) algorithm in a real cluster environment, called Gideon-II [11], which is the most powerful super computer at Hong Kong. We are assigned 8 physical nodes connected with 10Gbps high-speed intra-network. Each node has 8 2.45MHz-cores and 16GB of memory size. We deployed XEN 4.0 [22] on each node. Since there must be one core reserved for XEN hypervisor, we created 56(=8×7) VM instances (centos 5.2) on the 8 physical nodes. Three types of resource attributes (CPU rate, network bandwidth, and I/O disk bandwidth) will be split on demand according to our ODRA algorithm. Specifically, we make use of XEN's credit scheduler [16] to dynamically allocate various CPU rates (or capabilities) to the VM-instances. The network bandwidth and the disk reading/writing rate allocated to each user are both reshaped by linux network traffic controller (TC) [17] on demand at run-time.

In our experiment, each user task is constructed by multiple subtasks, each corresponding to various web services with heterogeneous workloads to process. The subtasks could also be data transmission via network or data read/write through disk. So, each subtask could be treated an execution dimension with a particular workload to process (e.g., number of float points and data to transmit) and a resource fraction (or processing ability) to allocate (e.g., CPU rate and network bandwidth).

By leveraging *ParallelColt* [18] (a library of matrix-computation), we implement 10 matrix computation programs in the form of web services (listed in Table 2), which can be further combined to construct more complex matrix problems. The number of subtasks per task is randomly set in [5, 15], and each subtask is a matrix computation selected from the 10 matrix computations. For example, the task $A_{M\times N}{\cdot}A_{N\times M})^k X_{M\times M}$=$B_{M\times M}$ are made up of three services: (1) matrix-multiplication: $C_{M\times M}$=$A_{M\times N}{\cdot}A_{N\times M}$; (2) matrix-power: $D_{M\times M}$=$C^k_{M\times M}$; (3) Least squares solution: $D_{M\times M}X_{M\times M}$=$B_{M\times M}$. The matrices in our experiments are randomly generated with the scales from 100×100 to 2500×2500, and their data sizes range from 192KB ({100×100}) to 115MB ({2500×2500}).

In our experiment, different tasks have different execution patterns. First, ten different matrix computations have various workloads, as shown in Table 2, where M refers to matrix size and $m(\in[10,20])$ is the exponent in the matrix-power computation. Second, each task execution involves three types of resources (CPU rate, network bandwidth and I/O disk bandwidth). For a particular task, the first subtask is reading one or more matrix data from disk drive. The second subtask could be some matrix computation like matrix product, decomposition, or others. As a matrix computation is finished, its output (a new matrix) will be transmitted to another VM instance through the network. The data transmission is also a kind subtask whose workload and capacity is data size to transmit and the network bandwidth controlled on demand. The last subtask of one task is storing the final matrix result into the disk drive. The CPU rate assigned to VMs is controlled by

TABLE 2: Workloads of 10 Matrix Operations (execution length per core, unit of measurement: seconds)

Matrix Scale	M-M-Multi.	QR-Decom.	Matrix-Power		M-V-Multi.	Frob.-Norm	Rank	Solve	Solve-Tran.	V-V-Multi.	Two-Norm
500	0.7	2.6	m=10	2.1	0.001	0.010	1.6	0.175	0.94	0.014	1.7
1000	11	12.7	m=20	55	0.003	0.011	8.9	1.25	7.25	0.021	9.55
1500	38	35.7	m=20	193.3	0.005	0.03	29.9	4.43	24.6	0.047	29.4
2000	99.3	78.8	m=10	396	0.006	0.043	67.8	10.2	57.2	0.097	68.2
2500	201	99.5	m=20	1015	0.017	0.111	132.6	18.7	109	0.141	136.6

XEN's credit scheduler [16]. Network bandwidth and disk I/O rate are both controlled within [10,300] Mb/s over NFS through Linux network traffic controller (TC). The CPU capacity of each multi-threaded program (e.g., matrix product, matrix power) is set to 8-cores (i.e., the maximum processing ability of one physical node), while that of any single-threaded program (such as matrix decomposition) is set to 1-core in that more resources cannot get further speedup on it.

We predict the workload of each matrix computation based on history for simplicity, where α and β are set to 0.7 and 2 respectively based on our characterization about prediction errors. In practice, one could use more accurate methods like multi-variate polynomial regression [14], whose margin of prediction errors is 7-10%.

Each task is associated with a budget, which is a key factor impacting task's resource allocation. The prices of the 10 matrix operations are set to 1,2,$\cdots$,10 with the decrease of their workloads. We evaluate our algorithm with different budgets assigned, which are simulated based on Formula (40), where $b(F_i)$ and θ (=0.5$\sim$3) refer to the price of the function F_i and a coefficient to tune users' various economic strengths respectively.

$$\tau' s\ budget = \theta \sum\nolimits_{i=1}^{K} b(F_i) \tag{40}$$

Upon receiving a task made up of multiple consecutive matrix computations, our designed algorithm is triggered to compute the optimal resource vector for it and perform resource isolation (e.g., notifying corresponding VMM to customize the CPU rates by credit scheduler).

We adopt three baseline results for comparison. The first one is the execution length (i.e., $f_I^{(*)}$) based on the ideal convex-optimal resource vector calculated by Theorem 1, with the assumption that the resource capacities are unbounded. We call it *ideal optimal time (IOT)*. The second one is the execution length (i.e., f_I^{*}) based on the practical optimal resource vector under the limited capacities in reality. We call it *practical optimal time (POT)*. The last one is one heuristic algorithm called *Load-Price-Ratio based Proportional-Share* (*LPRPS*). This heuristic is designed based on such an intuition: In one task, the subtasks with more workloads should be allocated with more resources, in order to minimize the total execution length. On the other hand, the subtasks with higher prices will cost higher than the ones with lower prices, thus the subtasks with lower prices are better to be allocated with more resources to make the payment more worthy. Hence, LPRPS tries to split the budgets among subtasks proportional to the Load-Price-Ratio (LPR). Based on Equation (3) and r_1:r_2:$\cdots$:r_R=$\frac{l_1}{b_1}$:$\frac{l_2}{b_2}$:$\cdots$:$\frac{l_R}{b_R}$, we could get LPRPS's resource vector as Formula (41), where κ=$\frac{B}{\sum_{i=1}^{R} l_i}$.

$$(r_1, r_2, \cdots r_R)^T = (\kappa\frac{l_1}{b_1}, \kappa\frac{l_2}{b_2}, \cdots, \kappa\frac{l_R}{b_R})^T \tag{41}$$

6.2 Experimental Results

There are two key metrics in our evaluation. The first one is called *execution stretch* (ES), aiming to evaluate task's execution performance. A task's ES is defined as its real execution length (with possibly erroneous workloads predicted) divided by its theoretically optimal execution length calculated based on its real workloads recorded after its execution. Smaller ES implies higher execution efficiency and ES being equal to 1 implies that the task's practical execution length reaches its theoretically optimal result. The other one is called *performance-payment ratio* (PPR), which is used to evaluate the effectiveness of user's payment. A task τ's PPR is defined in Formula (42), where τ's payment level is equal to $\frac{\tau' s\ final\ payment}{\tau' s\ budget}$. The smaller PPR, the higher performance with lower payment meanwhile, indicating higher satisfactory level.

$$PPR(\tau) = ES(\tau) \times (\tau' s\ payment\ level) \tag{42}$$

We first evaluate the impact of different budgets assigned to a single task to its execution performance and user's final payment. The task is computing $||(A^2_{2000\times2000} \times ((A_{2000\times2000} \times A_{2000\times1000}) \times \boldsymbol{v}_{1000})) \times \boldsymbol{v}_{1000}||_2$, where A and $\boldsymbol{v}$ means a matrix and a vector respectively. It is made up of 6 different matrix operations, including M-M-Multi., M-V-Multi., V-V-Multi., and so on. In Fig. 1 (a), we observe that the task's ES compared to its IOT increases linearly with the increase of the budgets assigned. This can be explained by Formula (3) or Equation (5). That is, higher budgets assigned will result in larger theoretically optimal resource amounts allocated, leading to a shorter IOT. However, the task has to be run atop the resources with limited capacities in practice, so we also compare task's real execution length to its theoretically optical value by taking capacity into account (i.e., POT). Fig. 1 (a) shows ES is always very close to 1 in this situation, which confirms that our solution is indeed able to optimize task's performance, based on user requirement and the load dynamics. In addition, the similar observation goes to the PPR metric, as shown in Fig. 1 (b), confirming the payment should also be satisfied by users.

We also compare the results in the situation with sufficient resources and the one with short supply respectively. Fig. 2 (a) shows the mean/lowest/highest values of the ES compared to IOT and POT respectively.

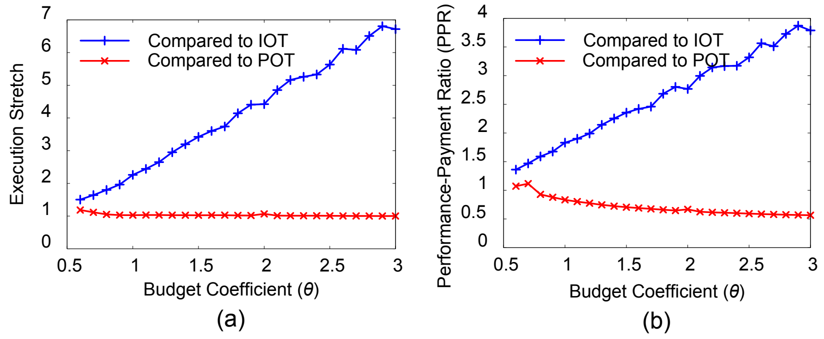

Fig. 1: Execution Performance under Different Payments. (a) Execution Stretch. (b) Performance-Payment Ratio

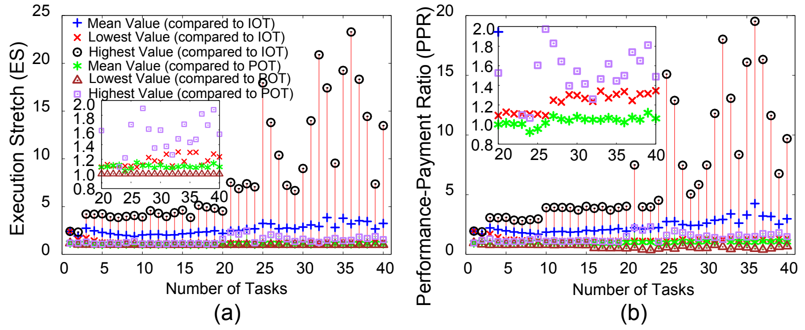

Fig. 2: Execution Performance with Different Number of Tasks. (a) Execution Stretch. (b) Performance-Payment Ratio

When the number of tasks is small (e.g. $\leq$10), the mean ES in both situations is always below 1.1, while the highest value of ES (worst situation) is up to 4. This is reasonable based on the following explanation. Note that the computation workloads among some subtasks (i.e., basic matrix operations) are largely different (e.g., between M-M-Multi. and V-V-Multi.), thus the resource amounts derived based on convex-optimization could be quite different. This will make the ideal optimal resource fractions of heavily-loaded subtasks be much bigger than the resource capacities (8-cores for the multi-threaded programs or 1-core for the single-threaded programs), such that subtasks cannot run in its ideal optimal states.

With further increasing number of tasks (from 10 to 40), the ES compared to IOT would increase notably, yet the ES compared to the POT still keeps pretty close to 1 (as observed in Fig. 2 (a)). In absolute terms, in comparison to POT, the mean value of ES can be limited down to about 1.1. This is attributed to the fact that more and more tasks cannot be assigned with the ideal optimal resource vectors (i.e., $\boldsymbol{r}^{(*)}$), while it can be assigned with the practical optimal resource vectors (i.e., $\boldsymbol{r}^{*}$) with regard to the limitation of the resource capacities. In fact, in such a situation with relatively short resource supply, task's practical optimal performance would also be degraded correspondingly, and ES$\approx$POT means that the tasks under our resource allocation run as efficiently as the practical optimal state with the capacity limitation. The similar observation goes with the PPR metric, as shown in Fig. 2 (b). When comparing to IOT, the PPR enhances with increasing number of tasks to process, yet it increases more slowly than that of ES shown in Fig. 2 (a), because users' payments are correspondingly reduced with the smaller resource amounts allocated.

We also evaluate the performance of the Algorithm 3 with the adaptive mechanism, which aims to further refine subtasks' resource allocation to adapt to the resource state changes at runtime. With the adaptive support, as some resources previously occupied by some tasks are released due to their completion, the optimal resource vectors of the other running tasks on the same node will be performed again. In Fig. 3 (a), we observe that the adaptive version of Algorithm 3 significantly outperforms the one without the adaptive support, via the mean ES. The mean ES with and without the adaptive mechanism is about 0.7 and 1.1 respectively. This confirms that task's execution would be impacted by the serious resource competition, while our adaptive mechanism could effectively solve the problem by adaptively reallocating the released resources at runtime. In addition, Fig. 3 (b) shows the PPR under the adaptive version of our algorithm. When there are 15+ tasks submitted (competitive situation), the mean value of PPR is much lower than that of the non-adaptive version by about 35% in general.

We also analyze the fairness of task's resource allocation based on Jain's fairness index [25] (Equation (43), where x_i is either ES or PPR.

$$F(\boldsymbol{x}) = \frac{(\sum_{i=1}^{n} x_i)^2}{n\sum_{i=1}^{n} x_i^2} \tag{43}$$

The fairness of ES (PPR) with respect to POT is much higher than the one with respect to IOT, as shown in Fig. 4 (a). This is because the basic matrix operations in our experiment are with largely different workloads

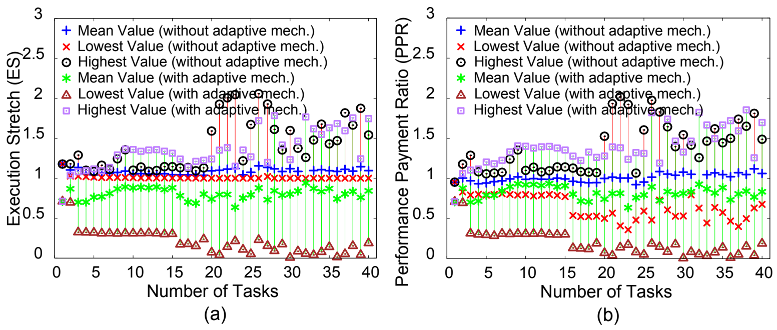

Fig. 3: Execution Performance without/with Adaptive Mechanism. (a) Execution Stretch. (b) PPR

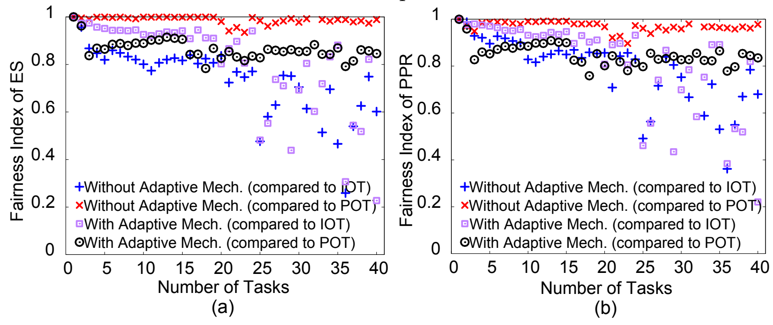

Fig. 4: Fairness of Task's Execution Performance. (a) Fairness Index of ES. (b) Fairness Index of PPR.

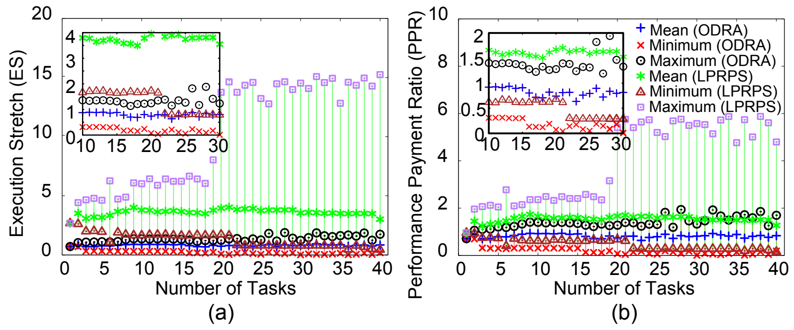

Fig. 5: Performance Comparison between ODRA and LPRPS. (a) Execution Stretch. (b) PPR.

(Table 2), which could easily make the resource fractions assigned at different dimensions are quite uneven. That is, the resource amounts expected at some dimensions may be extremely huge, finally succeeding the corresponding resource capacities to different degrees. Then, the degradation of the practical execution compared to IOT could be very arbitrary. However, we could still observe quite stable and highly fair treatment on task's resource allocation w.r.t. POT, i.e., when comparing to the execution length to the practical optimal state considering the capacity limitation.

Finally, we compare the execution performance of our algorithm to that of the Load-Price-Ratio based Proportional-Share (LPRPS) heuristic, as shown in Fig. 5 (Both are tested under the dynamic/adaptive resource allocation). We could clearly observe that LPRPS performs with much higher mean value of ES and PPR than that of our solution. Our ODRA algorithm can also effectively adapt to the competitive situation, i.e., the mean value and maximum value of the ES and PPR under ODRA stay quite stable even through there were over-many tasks submitted. In absolute terms, the mean values of ES and PPR under ODRA are about $\frac{1}{4}$ and $\frac{1}{2}$ of those under the LPRPS heuristic, which means ODRA outperforms the LPRPS heuristic significantly.

7 Related Work

Cloud resource allocation problem has been extensively studied for years, however, most of the existing work is strongly subject to the assumption with precise workload/hostload prediction. Usiao et al. [26] proposed a distributed load rebalancing method for distributed file systems in Clouds. Unlike the file system where data size is relatively easy to predict precisely, we have to deal with erroneous prediction issue in our computational cloud platform with multiple execution dimensions. Stillwell et al. [27] and Kuribayashi [28] both used greedy heuristics in resource allocation, and the processing ability and user requests in their simulations are assumed to be fixed and follow specific probability distributions (e.g., Gaussian distribution and exponential distribution). PACORA [29] is a performance-aware convex optimization model for resource allocation problem, assuming workload information could be known in advance. Goudarzi et al. [30] proposed a multi-dimensional SLA-based resource allocation for multi-tier applications, with the assumption that the average of user request and resource power are pre-known exactly. Jalaparti et al. [31] aims to optimize the resource allocation utilities between any two clients or client and provider. Their solutions have a strong assumption that the resource capacities are always large enough, while in our model, limited resource capacity is a key constraint, leading to a huge challenge especially in the bound analysis with prediction errors. Meng et al. [32] explicitly endeavored to maximize resource utilization by analyzing VM-pairs' compatibility in terms of the forecasted workload and estimated VM sizes. Their solution is able to approximately identify the compatibility of any pair of two VMs, but cannot resolve the situation with more than two VMs on the same machine. Wei et al. [33] formulated the Cloud resource allocation to be a binary Integer programming problem and solved it using an evolutionary method. A strong assumption in their work is the precise prediction of task's workloads on multiple execution dimensions.

In order to provide guaranteed service-level agreement (SLA), it is crucial to analyze the possible situation with inaccurately predicted information. Few works, however, fundamentally analyzed this issue for their approaches in the context of Cloud platforms. Mao's auto-scaling method [34] and Di's approach [35] took into account load prediction issue in Cloud systems, whereas they both handled a different objective that aims to minimize user payment with guaranteed task deadlines. Thus, the problem formulation is fairly distinct, so is the following solution. Wood et al. [36] adopted black-box and gray-box strategies for virtual machine migration, in order to alleviate hot spots based on statistical analysis. However, statistical analysis cannot be used to derive the bound of task execution performance at the worst case. In [37], AuYoung et al. evaluated the impact of inaccurate prediction for various utility-based scheduling approaches. They make use of simulation to analyze the working efficiency of First-Come-First-Serve (FCFS) scheduler and backfilling scheduler [38] with possible skewed estimate of application utility function and resource ability state. Although simulation work could confirm the fault tolerance ability to a certain extent, it cannot prove its effectiveness fundamentally.

There are also some exiting related works exploring how to make use of heuristics to adapt to the erroneous prediction problem. Kundu et al. [39] presented machine learning techniques to model the performance of a VM-hosted application as a function of resources allocated to VM. Their approach works well when being given a relatively precise load prediction (with 90th percentile of prediction errors being within [4.36%,29.17%]). However, it cannot keep guaranteed performance with relatively large load prediction errors. CloudScale [40] is a cloud system that automates fine-grained elastic resource scaling for multi-tenant cloud infrastructures. It employs online demand prediction and handles prediction errors to achieve adaptive resource allocation. In contract to this work, we theoretically derived the upper bound of the task execution length with different margins of prediction errors. Moreover, we improved the algorithm by making it adapt to dynamic changes of resource availability states, significantly enhancing the robustness.

Beyond the scope of Cloud computing, there are some existing works that discussed *robust convex optimization* problem, which is similar to our resource allocation issue with erroneous predictions. Robust convex optimization problem was first discussed in [41]. Some typical convex-optimization problems like linear programming and quadratic programming were studied in [41], under the assumption that some data are crude knowledge with uncertainty. Janak et al. [43] analyzed the robust optimization for Mixed-Integer Linear Programming (MILP) problem under uncertainty. Chaisiri et al. [44], [45] formulated the robust Cloud resource provisioning as a Stochastic Programming (SP) problem with uncertainty. In comparison to the above work, the resource allocation formulated in this paper is particular and largely different, due to the non-linear target function with multiple execution dimensions. To this end we designed LOAA algorithm to solve it with provably optimal output. Since LOAA is a specific algorithm, it needs particular analysis about its output with uncertain information.

8 Conclusion and Future Work

In this paper, we propose a novel algorithm (namely ODRA) which aims to minimize task execution length under a budget with possible prediction errors. We carefully derived the upper bound of task execution length for a practical situation with erroneous prediction of task workloads and resource availabilities. The derived bound of task execution length is very concise. To the best of our knowledge, this is the first paper to optimize the divisible-resource allocation with in-depth analysis on upper bound of task execution length under prediction errors. We also design a dynamic approach that adapts to the load dynamics over task execution progress. We evaluate the performance using a real cluster environment with composite web services. These services are of different execution patterns on multiple types of resources. Experiments show that task execution lengths with our ODRA solution are always close to their theoretically optimal results with resource capacity limitation. The mean values of Execution Stretch (ES) and Performance Payment Ratio (PPR) under ODRA are about $\frac{1}{4}$ and $\frac{1}{2}$ of those under LPRPS heuristic respectively, which means ODRA outperforms LPRPS heuristic significantly. In the future, we plan to improve system-wide performance by taking into account short supply situation. We will also implement more composite web services in addition to matrix computation, under the ODRA algorithm.

Acknowledgments

This research is supported by a Hong Kong UGC Special Equipment Grant (SEG HKU09), *Illinois-INRIA-ANL* Joint Laboratory on Petascale Computing, and also in part by the Advanced Scientific Computing Research Program, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

References

[1] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *Journ.al of the ACM (JACM)*, vol. 24, pp. 280–289, April 1977.

[2] F. Chang, J. Ren, and R. Viswanathan, "Optimal resource allocation in clouds," *Proc. third IEEE International Conference on Cloud Computing (Cloud'10)*, pp. 418–425, 2010.

[3] C. Jiang, C. Wang, X. Liu, and Y. Zhao, "A survey of job scheduling in grids," *Proc. of the joint nineth Asia-Pacific web and eighth int'l conf. on web-age information management conf. on Advances in data and web management (APWeb/WAIM'07)*, pp. 419–427, 2007

[4] J. E. Smith and R. Nair, *Virtual Machines: Versatile Platforms For Systems And Processes*. Morgan Kaufmann, 2005.

[5] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, "Enforcing performance isolation across virtual machines in xen," *Proc. seventh ACM/IFIP/USENIX Int'l Conf. on Middleware (Middleware'06)*, pp. 342–362, 2006.

[6] J. N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, and J. Owens, "Quantifying the performance isolation properties of virtualization systems," *Proc. ACM workshop on Experimental computer science (ExpCS'07)*, 2007.

[7] S. Chinni and R. Hiremane, "Virtual machine device queues," Virtualization Technology White Paper, Tech. Rep., 2007.

[8] T. Cucinotta, D. Giani, D. Faggioli, and F. Checconi, "Providing performance guarantees to virtual machines using real-time scheduling," *Proc. fifth aCM Workshop on Virtualization in High-Performance Cloud Computing (VHPC'10)*, 2010.

[9] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds : Managing performance interference effects for qos-aware clouds," *Proc. ACM European Conf. on Comp. Sys. (EuroSys'10)*, pp. 237–250, 2010.

[10] S. Di and C.-L. Wang, "Dynamic optimization of multi-attribute resource allocation in self-organizing clouds," *IEEE Trans. on Parallel and Distributed Systems (TPDS)*, 2012.

[11] Gideon-II Cluster: http://i.cs.hku.hk/∼clwang/Gideon-II.

[12] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2009.

[13] S. Di and C-L. Wang, "Minimization of Cloud Task Execution Length with Workload Prediction Errors," in *Proc. 20th High Performance Computing Conference (HiPC'13)*, 2013.

[14] L. Huang, J. Jia, B. Yu, B.-G. Chun, P. Maniatis, and M. Naik, "Predicting execution time of computer programs using sparse polynomial regression," *Proc. 24th Annual Conf. on Neural Information Processing Systems (NIPS'10)*, pp. 1–9, 2010.

[15] D. Sheng, D. Kondo, and W. Cirne, "Host Load Prediction in a Google Compute Cloud with a Bayesian Model," *IEEE/ACM 24th International Conference for High Performance Computing, Networking, Storage and Analysis (SC'12)*, pp. 21:1–21:11, 2012.

[16] Xen-credit-scheduler: on line at http://wiki.xensource.com/xen wiki/creditscheduler.

[17] W. Almesberger, "Linux network traffic control – implementation overview," 1999. [Online]. Available: http://diffserv.sourceforge.net/

[18] P. Wendykier and J. G. Nagy, "Parallel colt: A high-performance java library for scientific computing and image processing," *ACM Trans. Math. Softw.*, vol. 37, pp. 31:1–31:22, September 2010.

[19] S. Venugopal, J. Broberg, and R. Buyya, "OpenPEX: An Open Provisioning and EXecution System for Virtual Machines," Tech. Rep., CLOUDS-TR-2009-8, CLOUDS Laboratory, The University of Melbourne, Australia, Aug. 25, 2009.

[20] AWS economics: online at http://aws.amazon.com/economics.

[21] Google app engine (google code): on line at http://code.google.com/appengine/.

[22] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *Proc. nineteenth ACM symp. on Operating systems principles (SOSP'03)*, pp. 164–177, 2003.

[23] Google usage traces: http://code.google.com/p/googleclusterdata.

[24] S. Di, D. Kondo, and W. Cirne, "Characterization and comparison of cloud versus grid workloads," in *IEEE International Conference on Cluster Computing (Cluster'12)*, pp. 230–238, 2012.

[25] R. K. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modelling*, John Wiley & Sons, April 1991.

[26] H. Hsiao, H. Su, H. Shen and Y. Chao, "Load Rebalancing for Distributed File Systems in Clouds," *IEEE Trans. on Parallel and Distributed Systems (TPDS)*, 2012.

[27] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Gasanova, "Resource Allocation Algorithms for Virtualized Service Hosting Platforms," in *Journal of Parallel and Distributed Computing (JPDC)*, vol. 70, no. 9, pp. 962–974, 2010.

[28] S.-i. Kuribayashi, "Optimal joint multiple resource allocation method for cloud computing environments," *International Journal of Research and Reviews in Computer Science (IJRRCS)*, vol. 2, pp. 1–8, 2011.

[29] L. S. Bird and J. B. Smith, "Pacora: Performance aware convex optimization for resource allocation," *Proc. ACM third USENIX Workshop on Hot Topics in Parallelisation (HotPar'11) (Poster)*, 2011.

[30] H. Goudarzi and M. Pedram, "Multi-dimensional sla-based resource allocation for cloud computing systems," *Proc. first Int'l workshop on data center performance (DCPerf'11), held in conjunction with ICDCS'2011*, 2011.

[31] V. Jalaparti, G. Nguyen, G. Indranil, and C. Matthew, "Cloud resource allocation games," University of Illinois, Tech. Rep., 2010, Available: http://hdl.handle.net/2142/17427.

[32] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis, "Efficient resource provisioning in compute clouds via vm multiplexing," *Proc. of 7th IEEE International conf. on Autonomic computing (ICAC'10)*, pp. 11–20, 2010.

[33] G. Wei, A. Vasilakos, Y. Zheng, and N. Xiong, "A game-theoretic method of fair resource allocation for cloud computing services," *J. Supercomputing*, vol. 54, no. 2, pp. 252–269, 2009.

[34] M. Mao and M. Humphrey, "Auto-Scaling to Minimize Cost and Meet ApplicationDeadlines in Cloud Workflows," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*, 2011, pp. 49:1–49:12

[35] S. Di and C.-L. Wang, "Error-tolerant Resource Allocation and Payment Minimization for Cloud System," *Trans. on Parallel and Distributed Systems (TPDS)*, 2012.

[36] T. Wood, P.J. Shenoy, A. Venkataramani, and M.S. Yousif, "Black-box and Gray-box strategies for virtual machine migration," in *Proc. of the 4th USENIX conference on Networked systems design and implementation (NSDI'07)*, pp. 17–31, 2007.

[37] A. AuYoung, A. Vahdat, and A. C. Snoeren, "Evaluating the impact of inaccurate information in utility-based scheduling," *Proc. of ACM Conf. on High Performance Computing Networking, Storage and Analysis (SC'09)*, pp. 38:1–38:12, 2009.

[38] D. A. Lifka, "The anl/ibm sp scheduling system," *Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, pp. 295–303, 1995.

[39] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao, and K. Dutta, "Modeling Virtualized Applications using Machine Learning Techniques," *Proc. of the 8th ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments (VEE'12)*, pp. 3–14, 2012.

[40] Z. Shen, S. Subbiah, X. Gu and J. Wilkes, "CloudScale: elastic resource scaling for multi-tenant cloud systems," *Proc. of the 2nd ACM Symposium on Cloud Computing (SOCC'11)*, pp. 5:1–5:14, 2011.

[41] A. Ben-tal and A. Nemirovski, "Robust convex optimization," *Mathematics of Operations Research*, vol. 23, pp. 769–805, 1998.

[42] X. Lin, S.L. Janak, C.A. Floudas, "A new robust optimization approach for scheduling under uncertainty: : I. Bounded uncertainty," *Computers & Chemical Engineering*, vol. 28, no. 6-7, pp. 1069–1085, 2004.

[43] S.L. Janak, X. Lin and C.A. Floudas, "A new robust optimization approach for scheduling under uncertainty: II. Uncertainty with known probability distribution," *Computers & Chemical Engineering*, vol. 31, no. 3, pp. 171–195, 2007.

[44] S. Chaisiri, B.S. Lee, D. Niyato, "Robust Cloud Resource Provisioning for Cloud Computing Environments," *IEEE International Conference on Service-Oriented Computing and Applications (SOCA'10)*, pp. 1–8, 2010.

[45] S. Chaisiri, B.S. Lee, D. Niyato, "Optimization of Resource Provisioning Cost in Cloud Computing," *IEEE Transactions on Services Computing*, vol. 5, no. 2, pp. 164–177, 2012.

Sheng Di Sheng Di received his master degree (M.Phil) from Huazhong University of Science and Technology in 2007 and Ph.D degree from The University of Hong Kong in 2011. He is currently a post-doctor researcher at INRIA (France) and Argonne National Laboratory (USA). His research interest is optimization of distributed resource allocation and fault-tolerance. His background is mainly on the fundamental theoretical analysis and system implementation. Contact him at sdi@cs.hku.hk.

Cho-Li Wang Cho-Li Wang received his Ph.D. degree from University of Southern California in 1995. Prof. Wang's research interest includes multicore computing, software systems for Cluster and Grid computing, and virtualization techniques for Cloud computing. He serves on the editorial boards of several international journals, including IEEE Transactions on Computers (2006-2010), Journal of Information Science and Engineering, and Multiagent and Grid Systems. Contact him at clwang@cs.hku.hk.

Franck Cappello Franck Cappello has joined Argonne's Mathematics and Computer Science Division as senior computer scientist and project manager of research on resilience at the extreme scale. He also initiated and was the director of the Grid5000 project until 2009, a nationwide computer science platform for research in large-scale distributed systems. He leaded the XtremWeb (desktop grid) and MPICH-V (fault-tolerant MPI) projects and authored more than 100 papers in the domains of high performance programming, desktop grids, and fault tolerant MPI. He is now co-director of the INRIA-Illinois joint laboratory on PetaScale Computing and adjunct scientific director of ALADDIN/Grid'5000, the new four-year INRIA project aiming to sustain the Grid5000 infrastructure and to open it to cloud computing, service infrastructure, and the future internet research. He is one of the leaders of the Exascale effort through his participation to IESP and several other Exascale initiatives. He is an editorial board member of the international Journal on Grid Computing, Journal of Grid and Utility Computing, and Journal of Cluster Computing. He is a steering committee member of IEEE HPDC and IEEE/ACM CCGRID. Contact him at cappello@mcs.anl.gov.